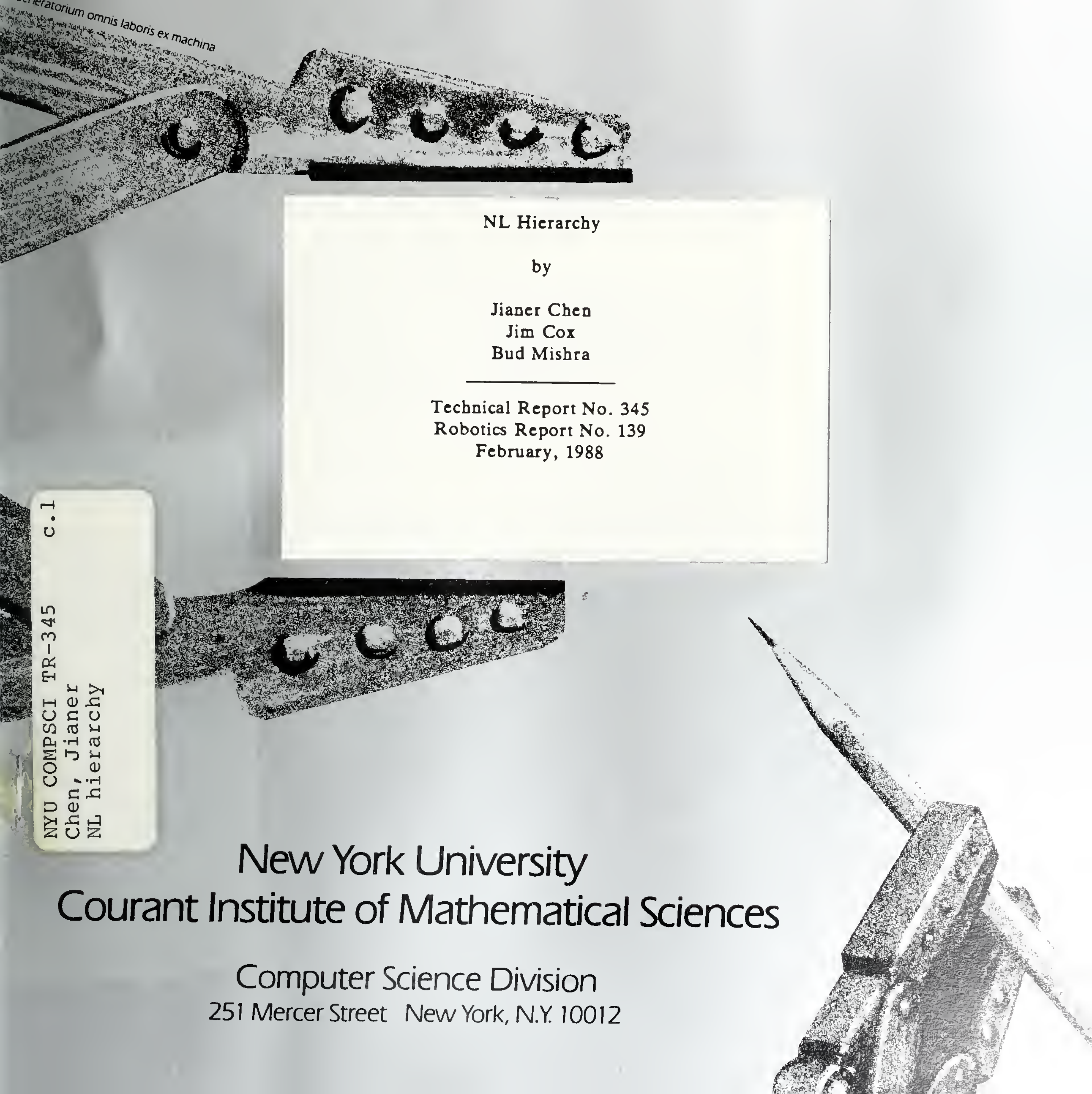
Robotics Research Technical Report

Generatorium omnis laboris ex machina



NL Hierarchy

by

Jianer Chen
Jim Cox
Bud Mishra

Technical Report No. 345
Robotics Report No. 139
February, 1988

NYU COMPSCI TR-345 c.1
Chen, Jianer
NL hierarchy

New York University
Courant Institute of Mathematical Sciences

Computer Science Division
251 Mercer Street New York, N.Y. 10012

NL Hierarchy

by

Jianer Chen
Jim Cox
Bud Mishra

Technical Report No. 345
Robotics Report No. 139
February, 1988

New York University
Dept. of Computer Science
Courant Institute of Mathematical Sciences
251 Mercer Street
New York, New York 10012

Work on this paper has been supported by Office of Naval Research Grant N00014-87-K-0129, National Science Foundation CER Grant DCR-83-20085, National Science Foundation Grant subcontract CMU-406349-55586, and by grants from the Digital Equipment Corporation and the IBM Corporation.

ABSTRACT

We investigate the power of a new class of nondeterministic space-bounded query machines and their relation to the PTIME-hierarchy. In particular, we demonstrate that the class 1NL, the languages accepted by a nondeterministic Turing Machine with a one-way read-only input head and $\log n$ *space, is a nontrivial class properly contained in NL. We give strong evidence that the language classes 1NL and DL are incomparable. We also introduce a new nondeterministic oracle model, compatible with time-bounded Turing-reducibility. The new model is significantly different from the one defined by Ruzzo, Simon and Tompa, and permits a nontrivial NL-hierarchy, i.e. this hierarchy does not collapse without a similar consequence for the PTIME-hierarchy. This is immediate from the result that* Σ_k *is exactly the* $2k^{\text{th}}$ *class in our NL-hierarchy.*

The new model also sheds some light on the question of relative power of "guessing" vs. "checking" in nondeterministic and alternating computation. This work was motivated by our effort to understand the power of "inductive counting," and various hierarchy-collapsing results based on it.

1 Introduction[1]

Neil Immerman showed that the complexity class for nondeterministic space $s(n)$ is closed under complementation, for $s(n)$ greater than or equal to $\log n$ [Immerman 1987]. As a corollary, he also settled a long-standing open problem by showing that the context sensitive languages are closed under complementation. In this paper, we study the power of *inductive counting*, the main proof technique used by Immerman.

In particular, we investigate a somewhat weaker class of languages, $1NL$, the class of languages accepted by a nondeterministic Turing Machine with a one-way read-only input head that uses only $\log n$ amount of space on its work tapes. We show that this language class is not closed under complementation. We also show that $1NL$ differs from CFL, NC_1, and DL and cannot contain either DL or NC_1. Using the characterization of $\log n$ space bounded complexity classes (DL and NL) by uniform polynomial size branching programs, we show that if $1NL \subset DL$ then nondeterministic space is equal to deterministic space for constructible space bound functions $s(n)$ greater than or equal to $\log n$. In particular we give a specific language $L \in 1NL$, which represents a reachability problem on regular graphs, and show that $L \in DL$ implies the equality of deterministic and nondeterministic space.

Thus $1NL$ is a non-trivial class of languages which is properly contained in NL and P, is not closed under complementation, is in some respects weaker than DL, and yet cannot be strictly weaker than DL without strong consequences for nondeterministic space. We then introduce a new oracle model, and show that an $1NL$ machine equipped with an $1NL$ oracle is precisely equal to NP under our new oracle model.

Borodin et. al. [Borodin et. al. 1987] have also investigated the power of inductive counting, and derived many interesting related results. The following result is stated in their paper:

$$NL^{NL} = NL,$$

where the oracle model is the one suggested by Ruzzo, Simon and Tompa; in this model the underlying nondeterministic machine must enter a deterministic phase to write on the query tape before submitting the query to the oracle [Ruzzo et. al. 1984]. Under our new model, which is compatible with time-bounded Turing reducibility, we allow the queries to be made nondeterministically as well as deterministically (The exact model will be discussed in the next section.) We show that, under the new model a result such as

---

[1] We assume familiarity with the basic results and terminologies of complexity theory, as in, for example, [Hopcroft et. al. 1979].

$$NL^{NL} = NL,$$

will imply that

$$NL = P = NP.$$

We also show that if inductive counting is strong enough to show that for each oracle A, the *complement* of $1NL^A$ is contained in NP^A then it follows that the *PTIME*-hierarchy collapses [Stockmeyer 1977]. Thus while inductive counting generalizes to the Ruzzo, Simon and Tompa model of space-bounded oracle machine, it may not generalize to our model.

To show the usefulness of our oracle model we introduce a hierarchy of languages

$$NL_1 = NL, \quad NL_2 = NL^{NL}, \quad NL_3 = NL^{NL^{NL}}, \quad \ldots$$

and show that it is very closely related to the *PTIME*-hierarchy. We then introduce some restrictions on our oracle model, while still allowing it to make non-deterministic queries. Under this second model of oracle machine (which can be thought of as a proof-generating machine) we can use inductive counting in certain cases. We can use inductive counting to show that for each k

$$NL^{\Sigma_k} = \text{co-}NL^{\Sigma_k}$$

where Σ_k is level k of the *PTIME* hierarchy. From this result we can show that an *NL* hierarchy built up from this model forms a refinement of the *PTIME*-hierarchy. A similar but not completely uniform refinement is built up, where levels alternate between *NL* and *DL*.

In our results we also indicate very sharply the relative power of "guessing" and "checking" in nondeterministic and alternating computations; specifically if the "prover" can guess a polynomial number (in the length of the input) of nondeterministic bits, the "verifier" can be strictly weaker than *PTIME* in power and we can still capture *NP* in one interaction.

2 1NL Class

We begin this section, which will primarily be concerned with the study of the class of languages accepted by one-way nondeterministic $\log n$ space

Turing machines, by introducing a new definition of oracle Turing machine. Ruzzo, Simon and Tompa's definition of space-bounded oracle machine have been concerned with restricting the power of the machine in making queries. This was done for several reasons. One of the main reasons for this approach was that if the machine was too powerful the model would not be useful in studying the "fine" structure of standard complexity classes. In this paper however, we intentionally introduce an extremely powerful notion of oracle machine in the hopes that it can shed some light on the power of "guessing" in nondeterministic computations.

An oracle Turing acceptor is a Turing acceptor that is equipped with a special write-only one-way oracle tape, and a distinguished state q_Q, called the query state and two distinguished query answer states q_{yes} and q_{no}, and operates as follows: during a computation with respect to an oracle set A, whenever the machine enters the query state in the next step of the computation the oracle tape is erased and the machine enters either of the two answer states. Let w be the nonblank contents of the oracle tape when it entered the query state. Then if $w \in A$ the machine enters q_{yes}, else it enters q_{no}.

An oracle machine is said to be $t(n)$ time-bounded (respectively $s(n)$ space-bounded) if on all accepted inputs of length n the machine accepts in at most $t(n)$ steps (respectively accepts in at most $s(n)$ space). If the machine operates deterministically it is said to be deterministic. We will now define our nondeterministic oracle machine.

Definition 2.1 (Nondeterministic Oracle Machine) A *nondeterministic oracle machine* (briefly, *NDO-Machine*) is an oracle machine with nondeterministic transitions subject to the following restriction: whenever the machine begins writing on a blank query tape it must enter a deterministic phase until the q_Q query state is entered, while retaining nondeterminism in the characters written to the query tape. More precisely let the finite control of the machine consist of tuples of the form $\langle q, c_1, \cdots, c_j, q_n, h_1, \cdots, h_j, w_1, \cdots, w_{j+1}\rangle$, where q specifies the current state, $c_1, \cdots, c_j$ the currently scanned characters, q_n the next state, $h_1, \cdots, h_j$ the head actions, and $w_1, \cdots, w_{j+1}$ the characters to be written, with w_{j+1} the character to be written to the query tape. Let τ_1 and τ_2 be two control tuples that are applicable to a configuration C during a query writing phase, so that $q, c_1, \cdots, c_j$ are the same in both. Then these two tuples are equal except for w_{j+1}, which specifies the character to be written to the oracle tape. □

Lemma 2.1 *Our definition of $O(t(n))$ time-bounded nondeterministic oracle machine is equivalent to the standard $O(t(n))$ time-bounded nondeterministic Turing reducibility.*

PROOF.

This equivalence is easily seen since any standard nondeterministic time-bounded query machine M can be simulated by a machine M' of our model by first writing the query to a special work tape. Then when M would enter the query state, M' first copies the contents of the special work tape to the query tape, erases the special tape and enters the query state. Now the action of the machine M' is completely deterministic during its query writing phase, and the time is increased by at most a small constant factor. ☐

Definition 2.2 (Deleted Configuration) A *deleted configuration* of a (oracle) Turing acceptor is a subset of an instantaneous description consisting of the state, input head position and currently scanned input character, and the contents and head positions of the work tapes. ☐

Note that the deleted configuration does not include the input tape contents or, in the case of an oracle machine, the query tape contents and head position.

Lemma 2.2 *A $s(n) = \Omega(\log n)$ space-bounded oracle machine accepts in time at most $2^{O(s(n))}$.*

PROOF.

The result is clearly true for deterministic space bounded oracle machines, so it suffices to show the result for nondeterministic machines.

First we show that the length of any query is bounded. Since the query tape is write-only, during a writing phase, each computation-step transition is dependent only on the deleted configuration. Since the machine uses at most space $s(n)$ it follows that the number of distinct deleted configurations is bounded by $2^{ks(n)}$ for some fixed positive constant k. Thus by our model if the machine ever repeats a deleted configuration during a query writing phase, the computation is looping and will continue writing to the oracle tape forever, without entering the query state.

Now the number of deleted configurations following a query (the state is either q_{yes} or q_{no}) is also similarly bounded. Thus by the usual argument there exists a fixed positive constant $c > k^2$ so that if an accepting computation takes more than $2^{cs(n)}$ steps, there must be a repeated configuration and a shorter accepting computation can be found. ☐

We note that the Ruzzo, Simon, and Tompa model of space-bounded oracle machine is a special case of our model. We now show an application of our model. We will use this idea later to build up an oracle hierarchy.

Theorem 2.3 *Let 1NL be the class of one-way NSPACE*[log *n*] *languages. Then*

$$NP \subseteq 1NL^{1NL}.$$

PROOF.
Let M be a nondeterministic polynomial-time Turing Machine,

$$\mathcal{L}(M) \in NP.$$

Here we assume without loss of generality and for technical convenience that each such machine is "clocked" so that on inputs of length n M never runs for more than n^k steps for some positive k. We define a language L_M as follows:

$$L_M = \left\{ \langle x, C_0\#C_1\# \cdots \#C_m \rangle \;\middle|\; \begin{array}{l} C_0\#C_1\# \cdots \#C_m \text{encodes an accepting} \\ \text{computation of } M \text{on input } x. \end{array} \right\}$$

where we assume some reasonable encoding of the machine's configurations and we assume without loss of generality that each C_i is of equal length in the encoding. Then it is easy to see that

$$\overline{L_M} \in 1NL,$$

since with logarithmic amount of space, a one way nondeterministic Turing Machine can guess the index of two configurations C_i and C_{i+1} $(1 \leq i < m)$ and verify that

$$C_i \not\vdash C_{i+1}.$$

To verify this it need only guess the position in which C_i and C_{i+1} are inconsistent with the transition function of M. Thus we can show that

$$\mathcal{L}(M) = \{x \mid M \text{ accepts } x\} \in 1NL^{1NL}.$$

First the base *1NL* machine guesses a valid computation of M on x, writes $\langle x, C_0\#C_1\# \cdots \#C_m \rangle$ on the oracle tape and queries *1NL* oracle $\overline{L_M}$. If the answer is no, then it has guessed the representation of a valid computation of M on input x and can accept x. ☐

We can show that an *NL* machine with *NL* oracle is equivalent in power to $1NL^{1NL}$.

Theorem 2.4

$$NL^{NL} = 1NL^{1NL} = 1NL^{DL} = NP.$$

PROOF.
First we observe that for each non-deterministic polynomial time ("clocked") acceptor M, L_M is in *DL*. This is because we can verify that for all i $C_i \vdash C_{i+1}$ using logarithmic space since we need only a counter for the position within a configuration. Thus to prove the theorem it suffices to show $NL^{NL} \subseteq NP$. Thus given an *NL* machine N using *NL* oracle A we can construct an *NP* machine M which accepts $L = \mathcal{L}(N^A)$ as follows. First we note that by the well known result using transitive closure, A is accepted by M_1 a deterministic *PTIME* acceptor. So M proceeds by simulating N except that it writes each query w that N makes to one of its work tapes. Then whenever N enters its query state, M simulates M_1 using the query w as its input. In time bounded by a fixed polynomial in the length of the query w. we can determine whether N would enter the yes or no state following the query. M then continues simulating N. The correctness of the simulation and the polynomial time bound follow immediately. □

Corollary 2.5

$$\begin{aligned} & NL^{NL} = NL \\ \Rightarrow \quad & NL \subseteq NP \subseteq NL \\ \Rightarrow \quad & NL = P = NP. \quad \square \end{aligned}$$

There has been some interest in recent years in seeing whether the number of queries made to an oracle enlarges the class of languages accepted (see [Kadin 1987]). Noting that the base *1NL* machine need make only one query to its oracle (*1NL* or *DL*) to capture *NP* we get

Corollary 2.6 *Let $1NL^{DL[1]}$ be the class of languages accepted by an 1NL machine with a DL oracle, making at most one query to its oracle. Then*

$$1NL^{DL[1]} = NL^{NL} = NP. \quad \square$$

We note the fact that a *DL* machine can recognize L_M leads to the following descriptive characterization of *NP*

Corollary 2.7 *A language L is in NP if and only if there is a DL predicate D (testable in deterministic log-space) and a polynomial p such that*

$$y \in L \iff \exists^p_x D(x,y),$$

where $\exists^p_x$ is the bounded existential quantification which stands for '$\exists_x |x| \leq p(|y|)$.' ☐

In fact since L_M can be recognized by two-head one-way finite automata which accepts a proper subset of *DL*[Yao et. al. 1978], the predicate D can be restricted to come from a class strictly weaker than P and we still get *NP*. This suggests that the power of guessing a polynomial number of nondeterministic bits, generating the proof, overwhelms the power of the deterministic part of the computation, the verification or proof checking.

Since *1NL* can be used to capture *NP*, it is natural to investigate the power of *1NL*. The rest of the section investigates that question. For example, can we show that *1NL* is closed under complementation? This question is easily answered in the negative.

Theorem 2.8 *1NL is not closed under complementation.*
PROOF.
Consider the following language:

$$L = \{x\#x : x \in \{0,1\}^*\}.$$

Now an *1NL* machine can recognize $\overline{L}$ in the obvious way, but we claim no *1NL* machine can recognize L.

Let M be an *1NL* machine that accepts L. We derive a contradiction. For each n and all inputs of length $2n+1$ the number of deleted configurations d of M is bounded by $(2n+1)^k$ for some constant k. For n large enough we can choose a set S of $m > (2n+1)^k$ distinct strings in L of length $2n+1$, where $S = \{x_i\#x_i\}$, $1 \leq i \leq m$. By assumption there exists an accepting computation Ψ_i of M for each $x_i\#x_i$. Choose one such Ψ_i for each i. Clearly M must read all the input in each Ψ_i. Thus we may choose a deleted configuration d_i from each Ψ_i where the input head is in position $n+1$, reading the '#' symbol. By the pigeon-hole principle there must exist two such configurations d_i and d_j $i \neq j$ so that $d_i = d_j$. From the fact that both Ψ_i and Ψ_j are accepting, we have that M must accept the string $x_i\#x_j$, which is clearly not in L, by the choice of S. ☐

From the proof of the preceding theorem we can conclude

Corollary 2.9 *The following hold:*

1. $DL \not\subseteq 1NL$.
2. $1NL \underset{\neq}{\subset} NL$.
3. $NC^1 \not\subseteq 1NL$.
4. $1NL \not\subseteq CFL$.

PROOF.
It is a simple matter to show that the language L in the previous proof is in DL. Since L is not in *1NL* it follows that $DL \not\subseteq 1NL$. Also since *1NL* is a subset of NL, and NL contains DL, it follows that *1NL* is a proper subset of NL. Given a reasonable binary encoding of $\{0, 1, \#\}$, it is straightforward to show $L \in NC^1$.

To see that there is a language in *1NL* that is not context-free, observe that the language $\{a^{[i]}b^{[i]}c^{[i]}\}$ on the three letter alphabet $\{a, b, c\}$ is not context-free but accepted by an *1NL* machine by using three counters. ☐

Since $1NL \underset{\neq}{\subset} NP$, it is obvious that there must exist an NP machine M, with $\mathcal{L}(M) \notin 1NL$, and thus $L_M \notin 1NL$. However, we can explicitly show such an M. Note that we have shown *1NL* cannot recognize the equality of two strings separated by a marker symbol. Let M be any NP machine accepting a non-sparse set (the number of strings in $\mathcal{L}(M)$ of length n exceeds the value of each fixed polynomial $q(n)$ for infinitely many n). A standard encoding of Hamiltonian Cycle is an example of such a language. We may use an argument almost identical to the proof that $L = \{x\#x\}$ is not in *1NL* to conclude that $L_M \notin 1NL$. The idea is that an *1NL* machine does not have enough distinct configurations available to it to distinguish which input string it has scanned upon reaching the first configuration. Hence it cannot distinguish a valid computation from an invalid one.

Corollary 2.10 *There is a non-deterministic polynomial time acceptor M such that*

$$L_M \notin 1NL. \quad \square$$

Now we have seen the limitations of *1NL*. A natural question to ask is whether *1NL* still captures the power of nondeterminism in an essential way. The next result shows that this is the case. If DL can simulate *1NL*, then nondeterministic space and deterministic space are equal in power (for all constructible $s(n) = \Omega(\log n)$).

To prove this result we first present a characterization of *NL* in terms of uniform polynomial size nondeterministic branching programs. Branching programs are models generalizing decision trees [Barrington 1986]. A branching program is a directed acyclic graph of local out-degrees two or zero and with a set of distinguished nodes: a START node (source, in-degree zero) and a set of *terminal* nodes (sinks, out-degree zero). All non-terminal nodes have out-degree two; the two out-going arcs labeled 0 and 1. A non-terminal node may be labeled by an input variable and corresponds to a *decision-node*; each terminal node is labeled 0 or 1, depending on whether it is a *rejecting* or *accepting* node, respectively.

We shall assume that the program is *leveled*, START is on level one and arcs are between nodes of adjacent levels only. If all non-terminal nodes are labeled by an input variable, we say that the program is *deterministic*; otherwise, the program is *nondeterministic*.

Each input Boolean string $x = x_1 x_2 \ldots x_n$ defines a set of paths from START to terminal nodes: the *computational paths* determined by x. The paths are defined as follows: start from the node START, after entering a nonterminal node N, if N is labeled by an input variable x_i, we proceed along the arc labeled by the value of x_i, otherwise (i.e., if N is not labeled by input variables), we proceed nondeterministically along any one of the two arcs. Each path ends at a terminal node. The string x is *accepted* by the branching program if x determines a path in the program which ends at a terminal node labeled 1.

The *size* of a branching program is the number of nodes. The *width* of the program is the maximum number of nodes on any level.

Let $P = \{P_i | i \geq 1\}$ where i is an integer, be a family of branching programs where P_n is a branching program for inputs of length n. We say P is *uniform* if there is a $\log n$ space bounded deterministic Turing machine M that on input 1^n, produces the encoding of P_n.

Definition 2.3 (UPNBP) The language class *UPNBP* is defined as follows: $L \in$ *UPNBP* if L is accepted by a uniform branching program family $\{P_i\}$, such that each P_i is nondeterministic. □

It is well known that *DL* is exactly the uniform polynomial size branching programs (see, for example, [Pudlak et. al. 1983] and [Cobham 1966]). A similar technique shows that

Theorem 2.11 *NL* = *UPNBP*.

PROOF.
Suppose $L \in UPNBP$. Lets say that M is the DL machine which generates $\{P_n\}$, the family accepting L. Then we can construct an NL machine which accepts L using the machine M as a subroutine to generate portions of the representation of P_n as needed, and guessing the accepting path, if one exists.

To demonstrate the converse, we sketch the construction of the uniform family $\{P_n\}$ which accepts $\mathcal{L}(M)$ for each NL machine M, so that the branching program is layered. For each n, P_n is a polynomial sized layered digraph with two rows for each time step of M's computation and a column for each deleted configuration d of M on inputs of length n. For each k, row $2k$ contains an edge from a node d to a node d' in row $2k+1$ if $d \vdash d'$ in a computation of M. Then for each node d in row $2k+1$, where the input head position encoded to be i, we label the node x_i and create an edge from d to a terminal node labeled 0 and an edge to the node d in row $2k+2$. These edges are labeled according to the input bit value in d. If the the input bit is 1 in d then the edge to the terminal node is labeled by 0, and the edge to d in the next row is labeled 1, else vice versa. Finally we include a start node in row 1 labeled by x_1 with appropriately labeled edges to each possible initial deleted configuration d (corresponding to the value of the first input bit). If a node has no successor it is labeled by 0 or 1 depending on whether the node represents a deleted configuration with an accepting state.

Now the outdegree of the even rows are at most 4. Thus by adding another $p(n)$ rows we get a nondeterministic branching program. The construction is very regular and can be carried out by deterministic $\log n$ space transducer. ☐

From the above we can show

Theorem 2.12 *$1NL \subset DL$ implies $DL = NL$.*
PROOF.
Let $L \in NL$. Then L is accepted by a $UPNBP$ family $\{P_n\}$. We can modify the DL machine M which generates this family to produce on input x an adjacency list representation of layered digraph G_x, with distinguished start node s and a target node t so that $x \in L$ if and only if t is reachable from s. That is, G_x represents computation paths for P_n on input x.

Now given such a layered graph (vertices divided into rows, all edges from row i to row $i+1$) we see that the reachability problem is in *1NL*, as an *1NL* machine can guess a path between two vertices, and the layering insures it need never rescan its input. If a *DL* machine can also solve this layered graph reachability problem, then we could use such a machine, together with M as a coroutine, to construct a *DL* acceptor for L. ☐

So we have exhibited a set $A \subset GAP$ in *1NL* which is complete for *NL* and thus cannot be in *DL* unless nondeterministic space and deterministic space are equal. We call this set A, the *layered graph reachability problem*.

Corollary 2.13 *There exists an NL-complete* $A \subset GAP$, *with* $A \in 1NL$ *so that if* $A \in DL$ *then for all constructible* $s(n) = \Omega(\log n)$ $NSPACE[s(n)] = DSPACE[s(n)]$.

PROOF.

From the *NL*-completeness of A, $A \in DL$ implies $DL = NL$. The usual padding arguments give the corollary. ☐

Thus despite its handicaps, *1NL* seems to be able to do some things *DL* cannot. Or perhaps can provide the proof that nondeterminism and determinism are equal with respect to space. And, importantly, *1NL* is a class provably weaker than P, unlike *DL* and *NL*.

3 Toward an NL Hierarchy

Since $NL^{NL} = NP$ this suggests that we can build up an *NL* hierarchy that captures the *PTIME* hierarchy. The following theorem suggests that this can be done.

Theorem 3.1 *Let* $NL_1 = NL$ *and let* $1NL_1 = 1NL$, *and for each positive integer* k *let* $NL_k = NL^{NL_{k-1}}$ *and* $1NL_k = 1NL^{1NL_{k-1}}$. *Then*

$$\Sigma_k \subseteq 1NL_{2k} \subseteq NL_{2k}$$

where Σ_k *is level* k *of the Meyer-Stockmeyer PTIME-hierarchy.*

PROOF.

The proof is by induction on k. The base $k = 1$ follows from the fact that $1NL^{1NL} = NL^{NL} = NP$. For the inductive step we assume the theorem holds for $k - 1$. So that $\Sigma_{k-1} \subseteq 1NL_{2k-2}$. Let us define the following language similar to L_M. Let L_{M^A} be defined as follows:

$$L_{M^A} = \left\{ \langle x, C_0\#C_1\#\cdots\#C_m\rangle \;\middle|\; \begin{array}{l} C_0\#C_1\#\cdots\#C_m \text{ encodes an} \\ \text{accepting computation of } M^A \\ \text{on input } x. \end{array} \right\}$$

that is, L_{M^A} is a reasonable encoding of valid computations of machine M with oracle A. We claim that

$$\overline{L_{M^A}} \in 1NL^A.$$

This is because, as with L_M, a $1NL$ machine equipped with an oracle A can guess an index i and verify that $C_i \not\vdash C_{i+1}$. That is, if it guesses that they are incompatible because C_i queries oracle A on word w and C_{i+1} represents an incorrect query answer, the $1NL$ query machine can write w to its query tape and verify this fact.

Now let $L = \mathcal{L}(M^A) \in \Sigma_k$ where $A \in \Sigma_{k-1}$. By the induction hypothesis $A \in 1NL_{2k-2}$ Now we have that $\overline{L_{M^A}}$ is in $1NL^A \subseteq 1NL^{1NL_{2k-2}} = 1NL_{2k-1}$. So L can be recognized by a $1NL$ oracle machine equipped with an oracle for $\overline{L_{M^A}}$. The inductive step follows immediately. The theorem then follows from the fact that $1NL_k \subseteq NL_k$. ☐

The next corollary observes that a collapse of the NL-hierarchy implies the collapse of the $PTIME$-hierarchy.

Corollary 3.2 *If there exists a positive integer k so that $NL_k = NL_{k+1}$ then the PTIME-hierarchy is contained in Σ_k.*

PROOF.
$NL_k = NL_{k+1}$ implies by the usual inductive argument that $\forall i NL_i \subseteq NL_k$. Since each NL query machine is also an NP query machine we have then $NL_k \subseteq \Sigma_k$. Since for each i $\Sigma_i \subseteq NL_{2i}$, we have immediately that the $PTIME$-hierarchy is contained in $NL_k \subseteq \Sigma_k$. ☐

If we consider the fact that L_{M^A} is in DL^A we can build up a hierarchy that is precisely equal to the Σ levels of the $PTIME$-hierarchy.

Theorem 3.3 *Let $(NL, DL)_1 = NL^{DL}$ and for each positive integer k let $(NL, DL)_k = NL^{DL^{(NL,DL)_{k-1}}}$. Then $(NL, DL)_k = \Sigma_k$.*

PROOF.
($\supseteq$) We show that $(NL, DL)_k \supseteq \Sigma_k$ by induction as in the previous theorem. The inductive step follows from the fact that an NL query machine can guess the computation of an $NP^{\Sigma_{k-1}}$ machine M on input x and write the guess to its query tape. A $DL^{\Sigma_{k-1}}$ machine can easily verify that the query represents an accepting computation, i.e. is in L_{M^A} where A is the Σ_k oracle. The inductive hypothesis immediately yields

$$\Sigma_k \subseteq NL^{DL^{\Sigma_{k-1}}} \subseteq NL^{DL^{(NL,DL)_{k-1}}} = (NL, DL)_k.$$

($\subseteq$) The proof is by induction on k. The base case follows from the fact that $NL^{DL} = NP$. For the inductive step assume that $(NL, DL)_{k-1} \subseteq \Sigma_{k-1}$. Let $L \in (NL, DL)_k$. Then $L = \mathcal{L}(M_1^{M_2^A})$ where M_1 is an NL query machine, M_2 is a DL query machine and A is an $(NL, DL)_{k-1} \subseteq \Sigma_{k-1}$ oracle. Now an NP query machine M with oracle A can recognize L. On input x M first simulates M_1, writing any query w that M_1 makes onto one of its work tapes. When M_1 enters its query state M simulates M_2^A taking the query as input, writing M_2's queries to its own query tape and employing its own oracle A to answer them. In this way it can answer M_1's query in time polynomial in the length of w and continue simulating M_1. The time of the entire simulation is bounded by a polynomial in the length of x. □

The key to showing that $(NL, DL)_k = \Sigma_k$ is that a DL query machine gives a definite answer in time polynomial in the length of the input. What we would like to do is show that the NL-hierarchy is a refinement of the polynomial hierarchy, i.e. that $NL_{2k} = \Sigma_k$. One way to do this would be to show via inductive counting or a similar technique that for each oracle A NL^A is closed under complement. Then one could use the standard technique of "oracle replacement" to simulate NL_{2k} by Σ_k. As we shall show using the techniques developed so far, such a result would imply the collapse of the *PTIME*-hierarchy. An alternate and weakened version showing that for "nice" complexity classes Λ co-$NL^{\Lambda} \subseteq NP^{\Lambda}$ would allow us to prove the same result using "oracle replacement." Even such a weakened result would imply the collapse of the *PTIME*-hierarchy. However, as we shall see in the next section, it is possible to restrict our oracle model so that we may employ inductive counting to complement certain oracle classes, and still build up an NL-hierarchy that forms a refinement of the *PTIME*-hierarchy.

The next theorem shows why the successful application of inductive counting in the naïve manner described above would imply the collapse of the *PTIME*-hierarchy.

Theorem 3.4 *If for each complexity class Λ co-$NL^{\Lambda} \subseteq NP^{\Lambda}$ then the PTIME-hierarchy collapses.*

PROOF.

Take $\Lambda = NL$. Then we have co-NP = co-$NL^{NL} \subseteq NP^{NL} = NP$. □

In the next section we show that by weakening our oracle model we can show the closure under complementation of some oracle classes and thereby obtain the result $NL_{2k} = \Sigma_k$.

4 NL Hierarchy

Definition 4.1 (A Restricted Oracle Model) We add the following restrictions to our oracle model. Let us assume without loss of generality, that all our languages are encoded in binary notation, that is, for each language A, $A \subseteq \{0,1\}^*$. We require that our oracle machines use only oracle sets on $\{0,1\}^*$ relative to the extended universe $\{0,1,\Diamond\}^*$. Thus if A is a language from the alphabet $\{0,1\}$ we may use the oracle A' which is the same as A relative to the larger alphabet; all strings containing $\Diamond$ are automatically in $\overline{A}$. Further if there are two control tuples τ_1 and τ_2 that differ only in the character written to the query tape, then there must be a third tuple identical to τ_1 except that the the symbol $\Diamond$ is written to the query tape If B is a base machine (resp. class) and A is an oracle set (resp. class) we write $B(R)^A$ to indicate that the oracle model used is the restricted oracle model. . □

Thus for each such oracle A any string with a $\Diamond$ in it is automatically in $\overline{A}$. And further each oracle machine that uses A as oracle has the property that if it can use nondeterminism in writing a query q, then it can write a query with the $\Diamond$ symbol in it. We call this symbol "the spoiler."

The idea behind the above restrictions is to insure that if our query machine ever uses nondeterminism in making a query, then it can write an ill-formed query; one that is bound to get a no answer. Further note that because under our oracle model the machine operates deterministically during query writing except for the choice of character written to the query tape, the machine can never "remember" the fact that it has written an ill-formed query. So its state and work tape contents are the same whether it has nondeterministically chosen to write the "spoiler" symbol or not.

In a sense our oracle machines can be thought of as theorem provers. They are interested in proving a series of theorems. Whenever the machine generates a query deterministically it can make use of a "no" answer. Whenever it uses nondeterminism during the query-writing phase, it is only interested in whether there exists a proof of a given theorem. Since it can always guess an incorrect or ill-formed proof (the "spoiler" insures this) it gets no real information from a "no" answer.

As an example of how our model *may* differ in power from our previous model consider the following example. Given an *NP* language L we define

$$NF(L) = \{1^{[n]} : \exists x |x| = n \wedge x \in \overline{L}\}.$$

That is, $1^{[n]}$ is in $NF(L)$ if and only if the nth segment of L is not full. Now it is a simple matter to see that $NF(L) \in NL^L$ under our previous model of oracle machine. The machine M on input $1^{[n]}$ just guesses a string of length n and writes it to its oracle tape and queries L. If the answer is "yes" we reject, if the answer is "no" we accept. This approach doesn't work with our new model since we would have no way of distinguishing whether we had guessed a string in $\overline{L}$ with respect to the original alphabet of L or whether our "no" answer was because we had written a "spoiler," that is posed an ill-formed question. In the analogy to theorem proving it is not apparent that we can easily determine whether there exists a non-proof of a given length with respect to some theorem.

Many of the results in the previous section are seen to hold under this new model. For example the computational definition of the Meyer-Stockmeyer *PTIME*-hierarchy remains unchanged under this new model, since as observed before, an $NP(R)$ oracle machine can write all its queries to the query tape deterministically. Also the results for NL query machines remain true for $NL(R)$ machines. In particular

Lemma 4.1 *Let* $NL(R)_1 = NL$ *and for each* k $NL(R)_k = NL(R)^{NL(R)_{k-1}}$. *Then* $\Sigma_k \subseteq NL(R)_{2k}$. ☐

The proof by induction goes through, since an $NL(R)$ oracle machine can still guess a NP^A computation, and accept if it gets a yes answer from its oracle. The possibility of writing an ill-formed query doesn't hurt, since we are only concerned with the existence of a yes answer. An $NL(R)^A$ machine can verify the correctness of such a guess, making all its queries deterministically.

Under this new (presumably) weakened model we can use inductive counting to complement $NL(R)$ based query classes if the oracle class is a sufficiently powerful and nice class. Each positive level of the polynomial hierarchy is such a class. In particular we can use the closure of Σ_k under polynomially bounded existential quantification to help in complementing an $NL(R)^{\Sigma_k}$ machine $M(R)^A$. We use Σ_k to answer questions of the form: "Given a deleted configuration d, is there a string w that the base $NL(R)$ machine M could be querying in d that is in the Σ_k oracle A?" So we can use this power of Σ_k as well as the fact that if the base machine uses nondeterminism in generating a query it can always get a "no" answer to show the following:

Lemma 4.2 *For each* k *co-*$NL(R)^{\Sigma_k} = NL(R)^{\Sigma_k}$.

PROOF.

The idea is to use inductive counting to find all reachable deleted configurations d of the $NL(R)$ query machine $M(R)^A$ on input x of length n, where $A \in \Sigma_k$, and accept if and only if none are accepting. We carry out this computation using an $NL(R)$ machine M' with Σ_k oracle. The inductive counting is carried out as in Immerman's result, with the exception that we regard an entire query writing phase as atomic. Specifically we call an *atomic step* of M one computation step in which the oracle tape remains blank, or a sequence of steps, starting with a step in which an initially blank query tape is written to, and ending in the first subsequent step in which the query state is entered. Then we compute all reachable deleted configurations in which the query tape is blank. We first show how to determine whether one deleted configuration of M yields another in one atomic step.

Suppose we are given two *partial configurations* d_1 and d_2, where each is an encoding of a deleted configuration, together with a bit telling whether the query tape is currently blank. If both d_1 and d_2 have blank query tape and no control tuple applicable to d_1 writes to the query tape, then M' can certainly determine whether $d_1 \vdash d_2$.

Suppose now that a control tuple applicable to d_1 writes to a currently blank query tape and d_2 is in the state q_{yes} or q_{no}. We now show how to use a Σ_k oracle to determine if d_1 yields d_2, in one atomic step of M. We can first attempt to have M' simulate the computation of M, using the query tape of M' to write the query w of M. We assume for the moment that we have access to oracle A. If the query w was written completely deterministically, we can determine if $d_1 \overset{*}{\vdash} d_2$ by an atomic step. Otherwise if any of the query was generated using nondeterminism, we can determine if $d_1 \overset{*}{\vdash} d_2$, excepting the state. This is because by assumption the entire atomic step is deterministic except for the characters written to the query tape.

In case nondeterminism was used in generating the query, we have several subcases. If the state of d_2 is q_{no}, then we can determine if $d_1 \overset{*}{\vdash} d_2$, irrespective of any oracle query answer M' gets. This is because by our model if a query is generated nondeterministically, we can always get a no answer from A by writing the "spoiler". If the state in d_2 is q_{yes}, then we may have gotten lucky and M' has already queried a string w that gets a "yes" answer from A.

Finally consider the case in which we are not so fortunate, that is, we know that $d_1 \overset{*}{\vdash} d_2$ if and only if M, started in the configuration $C(d_1)$

specified by d_1, can write a query string $w \in A$. Here we show how to use the closure of Σ_k under polynomially bounded existential quantification to answer the question. We essentially need an answer to the following question:

> Does there exist a partial computation (atomic step) of M on input x, taking $C(d_1)$ to a configuration C' in the query state, so that a string $w \in A$ is queried?

Call the oracle that answers this question B. B can be represented by a set of doubles $\langle x, d_1 \rangle$. To see that $B \in \Sigma_k$, note that an $NP^{\Sigma_{k-1}}$ machine can nondeterministically guess the partial computation of M, write its query w to a work tape, and then perform an $NP^{\Sigma_{k-1}}$ computation that verifies $w \in A$.

Finally since we have to make use of both A and B, we encode A and B as $C = A \oplus B = \{0x | x \in A\} \bigcup \{1x | x \in B\}$.

Now we can carry out inductive counting with the $NL(R)^{\Sigma_k}$ machine $M'(R)^C$. Let the run time of M be bounded by the polynomial $p(n)$. As in Immermans technique suppose inductively that we have computed N_{i-1} the number of partial configurations reachable in at most $i-1$ of the above atomic steps of M. We show how to compute N_i.

$M'(R)^C$ can lexicographically iterate through all partial configurations d_t. For each such target configuration d_t, we also iterate through each partial configuration d, see if we can guess a computation of M of at most $i-1$ atomic steps to our intermediate target d. If so we decrement a counter initialized to N_{i-1}. Then we determine if d yields d_t in an atomic step as described. If so, we increment our counter for N_i and generate the lexicographical successor to the target d_t. Else we generate the successor to the intermediate target d. If we guess computations to N_{i-1} such intermediate targets d, and none yield d_t, we can continue with the next target. Thus given N_{i-1} we can successfully compute N_i and carry out the inductive counting.

After successfully computing $N_{p(n)}$ we can determine if $M(R)^A$ rejects x. The work storage uses just enough space store several counters that count up to $p(n)$, and several partial configurations at once. The work storage is thus seen to be at most $c \log n$ for some positive constant c. ☐

With the previous lemma in hand it is now a straightforward matter to use "oracle replacement" to show

Theorem 4.3 *The* $NL(R)$*-hierarchy is a refinement of the PTIME-hierarchy, that is, for each* k

$$NL(R)_{2k} = \Sigma_k.$$

PROOF.
We show by induction on k that $NP^{\Sigma_{k-1}}$ can simulate $NL(R)_{2k}$. Let $L \in NL(R)_{2k}$. Let $L = \mathcal{L}(M(R)^A)$, where M is an $NL(R)$ query machine and we can assume by the inductive hypothesis that $A \in NL(R)_{2k-1}$ is accepted by an $NL(R)$ query machine M_1 with Σ_{k-1} oracle B. Then by the previous theorem there exists an $NL(R)$ machine M_2 with Σ_{k-1} oracle C which accepts $\overline{A}$.

Then an NP machine with Σ_{k-1} oracle $B \oplus C$ can accept L as follows. It first simulates M on input x, writing each query to a worktape. To get the answer to a query w it nondeterministically simulates either M_2 or M_3, using oracle $B \oplus C$, and continues the simulation just in the case that w has been accepted by either M_2 or M_3. In this case it has a definite answer to the query w and can continue simulating M. Thus $L \in \Sigma_k$. ☐

Unfortunately it seems that we cannot use inductive counting to complement "even" levels of the $NL(R)$-hierarchy or we could collapse the polynomial hierarchy. Also it seems that $NL(R)$ query machines are different in power than P query machines because the $NL(R)$ query machines can use nondeterminism to guess polynomially many bits, where the P machine cannot. This leads to seemingly anomalous results such as the following: An $NL(R)$ machine with Σ_k oracle seems strictly weaker than a P machine with a Σ_k oracle, yet an $NL(R)$ machine with Δ_k oracle seems strictly stronger than a P machine with Δ_k oracle.

Theorem 4.4 For each k,

$$NL(R)^{\Sigma_k} \subseteq P^{\Sigma_k}$$

but

$$NL(R)^{P^{\Sigma_k}} \subseteq P^{P^{\Sigma_k}} = P^{\Sigma_k}$$

implies the collapse of the $PTIME$-hierarchy.

PROOF.
It is straightforward, but detailed to modify the proof of the closure under complementation of $NL(R)^{\Sigma_k}$ to show that $NL(R)^{\Sigma_k} \subseteq P^{\Sigma_k}$ by systematically enumerating all deleted configurations, and avoiding the use of

nondeterminism. Now

$$NP^{\Sigma_k} = NL(R)^{P^{\Sigma_k}} \subseteq P^{\Sigma_k}.$$

says that the *PTIME*-hierarchy collapses to the Δ_{k+1} level. ☐

5 Conclusion and Open Problems

In this paper we have introduced a new and more powerful model of nondeterministic space-bounded oracle machines that allows the construction of nontrivial space-bounded oracle hierarchies. The previous space hierarchies (under the Ruzzo, et. al. model) have now been shown to collapse by Immerman's result [Immerman 1987]. We have shown that if our *NL*(R) hierarchy collapses, so does the *PTIME* hierarchy. It is interesting to ask what is the precise relationship of the *NL* hierarchy to the *PTIME*-hierarchy. Is $NL(R)^{\Sigma_k} = P^{\Sigma_k}$? For example, we conjecture that our first definition of oracle machine, without the restriction on negative answers, defines an *NL*-hierarchy that is a refinement of the *PTIME*-hierarchy. Can this be proven? In general, can it be shown that the model of *NDO-Machine* is the same as the restricted model of *NDO-Machine*, i.e. is $NL(R)^A = NL^A$ and is $NL(R)_k = NL_k$? What general results can be proven about nondeterministic space-bounded oracle machines and classes? What conditions are needed for techniques such as inductive counting to relativize? In this case they seem to relativize to a very powerful oracle model subject to certain restrictions. Can we relax these restrictions?

We have demonstrated that under our unrestricted model $NP = 1NL^{1NL}$. This is interesting since the *1NL* class is quite simple, and strictly weaker than *P*. Recently inductive counting has been used to show that several of the lower level complexity classes are closed under complementation [Borodin et. al. 1987]. Thus it is now known that both *NL*, and SAC_1 = *LOGCFL* are closed under complementation, leading some to conjecture that they are equal. Since *1NL* is not closed under complementation, can studying this class lead to insight into inductive counting and collapsing hierarchies? In particular, Immerman has given a descriptive characterization of *NL* as *FO*+ *TC* (first order logic with a transitive closure operator), and shown why this class is closed under complementation. Can we give a similar descriptive characterization of *1NL* and use this to gain more insight into the general conditions necessary for closure under complementation of complexity classes?

References

[Borodin et. al. 1987] A. Borodin, S. Cook, P. Dymond, W. Ruzzo, and M. Tompa, "*Two Applications of Complementation via Inductive Counting*," **Manuscripts**, September, 1987.

[Barrington 1986] D. Barrington, "*Bounded-Width Polynomial Size Branching Programs Recognize Exactly Those Languages in* NC^1," **Proc. Symp. on Theory of Computing 18**, 1986, pp.1-5.

[Cobham 1966] A. Cobham, "*The Recognition Problem for the Set of Perfect Squares*," **Research Paper RC-1704, IBM Thomas J. Watson Research Center**, Yorktown Hieghts, NY, April 1966.

[Hopcroft et. al. 1979] J. Hopcroft and J. Ullman, **Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, Mass**, 1979.

[Immerman 1987] N. Immerman, "*Nondeterministic Space is Closed Under Complement*," Technical Report 552, Yale University, July 1987.

[Kadin 1987] J. Kadin, "*Is One NP Question as Powerful as Two*," **Technical Report, Cornell University 87-842**, June 1987.

[Pudlak et. al. 1983] P. Pudlak and S. Zak, "*Space Complexity of Computations*," **Technical Report, University of Prague**, 1983.

[Ruzzo et. al. 1984] W. Ruzzo, J. Simon, and M.Tompa, "*Space-Bounded Hierarchies and Probabilistic Computations*," **Journal of Computer and System Sciences 28**, 1984, pp.216-230.

[Stockmeyer 1977] L. Stockmeyer, "*The Polynomial-Time Hierarchy*," **Theoretical Computer Science 3**, 1977, pp.1-22.

[Yao et. al. 1978] A. Yao and R. Rivest, "$K+1$ *Heads are Better than* K," **Journal of ACM 25**, 1978, pp.337-340.

NYU COMPSCI TR-345 c.1
Chen, Jianer
NL hierarchy

NYU COMPSCI TR-345 c.1
Chen, Jianer
NL hierarchy

This book may be kept JUN 10 1988

FOURTEEN DAYS

A fine will be charged for each day the book is kept overtime.

GAYLORD 142 PRINTED IN U.S.A.